Cluster Computing (2020) 23:2517–2531
https://doi.org/10.1007/s10586-019-03025-w

Private computation of the Schulze voting method over the cloud

Vijay Kumar Yadav[1] · Anshul Anand[1] · Shekhar Verma[1] · S. Venkatesan[1]

Received: 28 September 2019 / Revised: 13 November 2019 / Accepted: 2 December 2019 / Published online: 9 December 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract
In this work, we propose an algorithm that computes the Schulze voting method privately and finds the winner of the candidate without revealing the preferences of the voter. Users often outsource data to the cloud to get the result of an intended computation. Many a time, the user would like to keep the data and the result of the delegated computation private due to its sensitive nature. It is possible using data analytics to extract private information from a person. Hence, there is a need to perform computation on encrypted data, which can protect from a leak of private information. Homomorphic encryption (HE), allows computation on encrypted data. HE scheme takes input data in encrypted form and produces output in encrypted form. This encrypted output can not be decrypted without the private key. The Schulze method involves computation of a more complex function known as *strength of strongest path*. This is challenging to implement privately because it requires the evaluation of several functions over the ciphertexts. We use the Levelled-Brakerski–Gentry–Vaikuntanathan (BGV) fully homomorphic encryption (FHE) scheme to privately compute the strongest paths in a weighted graph using a modified image result of the Floyd–Warshall algorithm. We evaluated our proposed algorithm using an FHE library HElib. Besides, we implemented it for various parameters like time and number of levels in the modulus chain and also evaluated the size of the public key and secret key used for encryption and decryption. From the implementation results, we found that if we increase the number of levels, then the computation and communication complexity will also increase. Therefore, for efficient computation, we need to choose the optimal level.

Keywords Homomorphic encryption · Fully homomorphic encryption · Cloud computing · Schulze voting method · Ring learning with error

1 Introduction

Recently, there is a trend of delegating or outsourcing of digital data to the cloud platforms for the computation [1]. On the sidelines of this development, we can see an increasing concern about a user's privacy. Extracting sensitive information from a collection of useful looking data is also now a possibility [2–4]. Hence, there is a dichotomy where the user wants to perform cloud-based or server-based computing due to computational constraints and also keep its sensitive data and the result of the computation as private to prevent the breach of privacy. For example, a scientific lab might want to outsource the computation of fingerprint matching but would not want to reveal the sensitive biometric information to the cloud server as the cloud server might not be a complete trusted party. There is the addition of a third dimension where apart from computation and privacy, we also want absolute transparency to the whole calculation as in the case of open source development where the purpose of the work and certain other information is out in the open. HE allows us to compute on encrypted content. For example, in Fig.1, we can see that user first encrypts the readable data and sends it to the cloud in the unreadable or scrambled form. Since the encryption scheme HE is used, the cloud provider is able to perform computation on it in the form of Eval() function. The result obtained after Eval() is also unreadable and the user gets the result of the operation after decryption [5]. Till 2008, HE did not have any scheme that computes addition and multiplication on any number of times simultaneously. Hence, they were limited to the amount of computation they could perform. It is now possible to

✉ Vijay Kumar Yadav
pcl2014002@iiita.ac.in

[1] Indian Institute of Information Technology, Allahabad, India

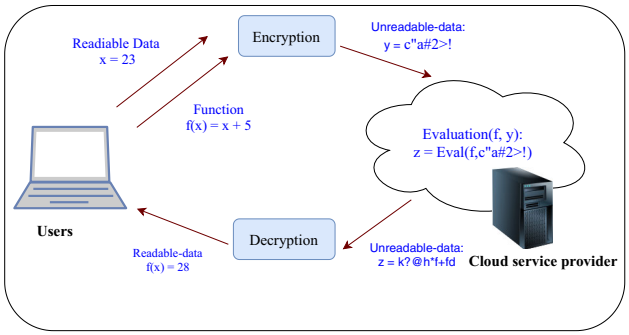

Fig. 1 The working of cloud computing using homomorphic encryption

compute complex operations like comparison, maximum, and minimum of two encrypted numbers [6].

Previously, lots of researchers use cloud-based software to improve their research. At present, a lot of problems and relations are being modeled as graphs [7]. The solution to graph problems like the shortest distance between the nodes etc. give the desired information of these graphs. Hence, there is a need to compute the graph algorithms privately. In this work, we intend to provide an algorithm for private computation of widest path problem and their implementation. Using this algorithm, we evaluate Schulze voting method privately. Schulze voting method is an essential voting method which is used by some of the organizations like Gentoo, Debian, political parties, and Wikimedia. In the Schulze voting method, the voting preferences of the electors are provided to the cloud server for evaluation of winner of the candidate without disclosing preferences of the elector. If the cloud server collides with the candidates and revealing preferences of the elector, then this might become sensitive because the candidates might against people who have not given them a voting choice. In this work, we have used the Leveled-BGV FHE scheme for the evaluation of Schulze voting method to perform private computation.

1.1 Problem definition

Graph modeling algorithm is extensively used to extract sensitive information. Since the data involved can be sensitive, we need to have private computation methods for graph algorithms. The motivation is to provide a solution to a graph problem with a real-world application. Schulze Voting method is a preferred voting method where users have to provide their order of preference. The preference of a voter must be kept private [8]. Currently homomorphically encryption has only been used to computes conventional election [9]. *Conventional voting* is known as *past the post*, in which the result is obtained by the simple addition of votes to check who has the higher number of votes [10]. Till now, there has been no work showing the construction and implementation for Schulze voting method, because it requires computation of complex operations like comparison, finding the maximum of two numbers, finding the minimum of two numbers, and binary addition, etc. in a private manner. Also, it requires designing and constructing a secure algorithm for computation of the strongest path is directed weighted graphs with encrypted weights and has been an unexplored area till now. Since, Schulze voting is used by a lot of organizations across the globe. There can be privacy concerns about revealing the voting preferences of a department or a team. Hence, we are motivated to provide a private computing method for Schulze voting.

- Till now, HElib library provides an open-source implementation of Leveled-BGV FHE. Besides, the implementation for the private computation of comparison, binary carry adder, finding maximum, and finding the minimum. Till now, there is no open-source library for the private computation of strongest path strengths in the graph.
- The implementation of private computation of Schulze voting method requires computation of strongest path strengths, which is very difficult.
- We aim to solve the private computation of voting preferences in Schulze voting method using Leveled-BGV.

1.2 Contributions

The Schulze voting method is a Condorcet voting method which requires computation of strongest paths in a weighted graph to find out the winner. It is different than the private computation of traditional voting schemes for election. In traditional voting, the employee first past the post system, which requires simple increment of the vote count and check the maximum number of votes in the end. Our approach leverages the principles of using Leveled-FHE scheme to implement computational circuits so that the strongest paths in a weighted graph can be computed privately. In summary, the significant contributions are:

– We describe the private computation for deciding the winner of the participating candidates in the Schulze Voting method [11].
– We propose an algorithm that computes the strongest path in the preferences weighted graph privately. Using our proposed algorithm, we evaluate Schulze Voting method privately and find the winner of the candidate without revealing the preferences of the voters.
– In order to realize the algorithm, we implement the encrypted evaluation of comparison, minimum and

maximum operations using the HElib library. We have implemented the aforementioned primitives using HElib which employs the BGV Leveled FHE Scheme and provide an open-source implementation available at[1] to the source code to the research community.

– We evaluate our algorithm using an HElib implementation for various parameters like time and number of levels that are required for the evaluation of Schulze Voting method privately. Besides, evaluate the size of the public key and secret key used for decryption and encryption for the different number of levels in the modulus chain.

The rest of the work has been set up in the following manner. In Sect. 2, we will describe the related work of HE and FHE and their application that has taken place in the domain of private computation. In Sect. 3, we will describe the pre-requisite of our proposed work. In Sect. 4, we will discuss Leveled-BGV FHE scheme. We will discuss the Schulze voting method in Sect. 5. Our proposed scheme private computation of Schulze Voting method we will discuss in Sect. 6. We will discuss the results and implementation details of our proposed scheme in Sect. 7. We will discuss conclusion of our work in Sect. 8.

2 Related work

Gentry's FHE scheme provides a significant growth in the field of HE [6]. Problem with the Gentry's scheme is that when the depth of the circuit increases concerning the noise in the ciphertext also increases. Due to increases the noise in the ciphertext it creates the decryption failure problem. Gentry proposed a noise management technique called bootstrapping, which reduces noise in the ciphertext. But, this bootstrapping scheme is inefficient and high communication overhead. Smart and Vercauteren worked together to improve upon Gentry's scheme by reducing the size of the ciphertext and the key size [12]. In [13], introduced an FHE scheme over integers using only elementary modular arithmetic, and their scheme's security is depending on the hardness of the Approximate-GCD problem. Brakerski and Vaikuntanathan gave another scheme in [14], whose hardness is based solely on the LWE assumption, and the security of the scheme is based on "shortest vector problem" on lattices. Brakerski et al. introduced an FHE scheme known as BGV [15]. This BGV scheme does not require the need for bootstrapping, and it is a leveled scheme capable of evaluating polynomial of arbitrary sizes. The BGV scheme provides the option to use either Learning With Error (LWE) or Ring-LWE assumptions. Lopez et al. proposed a multiple key-based FHE scheme using the NTRU cryptosystem in [16]. Gentry et al. in [17] also proposed an FHE scheme popularly known as the GSW. The GSW scheme is based on the "approximate eigenvector method" and its security is based on the LWE assumption. Recently, Chillotti et al. in [18] improved the work of GSW by using the mathematical structure of Torus to construct a quick FHE scheme.

Currently, HElib [19] is the most important and widely used FHE library. The implementation of the HElib library given by Gentry–Halevi–Smart in which they have focused on the optimizations of HElib library. Besides, the ciphertext packing techniques are provided by Smart–Vercauteren for improving the performance of HElib library. Halevi and Shoup describe in [20] the algorithms and hardware used in the implementation of HElib. Another popular library for FHE given by Microsoft is *SEAL*. Ducas et al. also given *FHEW* library [21] for FHE. There are many critical applications of FHE and HE has been proposed recently [22, 23].

We have compared the related work of existing voting schemes in Table 1. In this table, the existing schemes [9] and [10] working on the conventional voting scheme, whereas the proposed scheme is working on the preferential voting scheme. In [9], they have discussed the verification of tempering electronic voting machine (EVM) using Homomorphic ElGamal Encryption (HEE). The time and space complexity of this scheme is $\mathcal{O}(5N\log_2 N)$ and $|4\mathcal{G}_q|$, respectively. Where N and $|\mathcal{G}_q|$ denotes the number of participating candidates in the election and size of group element over q, respectively. In [10], they have discussed the conventional voting using HEE. The time and space complexity of this scheme is $\mathcal{O}(\sqrt{M}^{l-1})$ and $\mathcal{O}(2MV\mathcal{G}_q)$ respectively. Where l denotes the number of votes, M denotes the number of voters, and V denotes the number of the authorized parties who evaluate the votes. Our proposed scheme uses the FHE technique for the evaluation of preferential voting. The detailed description of the time and space complexity of the proposed scheme is given in result and implementation Sect. 7. From this table, it is clear that the space complexity of the proposed scheme only depends on the number of candidates participating in the election and the security parameter λ. Whereas existing conventional voting scheme's space complexity depends on the number of voters participating in the election, which is very high as compared to the proposed scheme. Therefore, our scheme gives the best results in terms of the ciphertext context. Our proposed scheme uses lattice-based cryptography, which makes our scheme quantum-resistant, whereas the quantum computer can easily attack the existing voting methods.

[1] https://github.com/anshul92101/FHE-Schulze/.

Table 1 Comparison of related work of existing voting schemes vs. our scheme

Paper	[9]	[10]	Our
Proposed	Secure and verifiable EVM	Receipt free voting	Preferential voting
Time complexity	$\mathcal{O}(5N\log_2 N)$	$\mathcal{O}(\sqrt{N}^{l-1})$	$\mathcal{O}(n.N^3 + N.(N-1)\lambda.L^3)$ or $\mathcal{O}(nN^3 + N.(N-1).\lambda^2)$
Space complexity	$\mathcal{O}(\lvert 4\mathcal{G}_q\rvert)$	$\mathcal{O}(2MV\mathcal{G}_q)$	$\mathcal{O}(N.(N-1).\lambda)$
Used technique	HEE	HEE	FHE
Quantum resistant	No	No	Yes

3 Background

3.1 Basic notation

The following notations are used throughout the paper. We denote $e \leftarrow \chi$ as a generation of an error using discrete Gaussian distribution. Whereas, $e \leftarrow \chi^{\mathbf{N}}$ denotes the generation of **N** error using discrete Gaussian distribution. q is a prime number and an integer d is power of two, which satisfy the condition $q \equiv 1 \ mod\ 2d$. Besides, we choose polynomial ring $R = \mathbb{Z}[x]/(x^d + 1)$ of degree $d-1$, and the reduction of polynomial coefficient in the range of $(-q/2, q/2]$ can be denoted as $R_q = R/qR$. We represent $\mathbb{Z}$ as integer, $\mathbb{Z}_q$ as a finite field over q, and $\mathbb{Z}_q^n$ as a set of n elements over finite field q. "." denotes either Euclidean product of two polynomials or Euclidean dot product of two vectors. |.| denotes the Euclidean norm.

3.2 LWE and ring-LWE

LWE problem was initiated in [24]. In recent years, LWE has become one of the most popular and useful schemes for creating powerful quantum-resistant cryptographic constructions. All the cryptographic constructions using the LWE problem are based on either worst-case hardness or average-case hardness assumption. The LWE problem says that to recover a secret vector $s = \{s_1, s_2, \ldots, s_n\} \in \mathbb{Z}_q^n$ from a given sequence of 'approximate' n random linear equations is a hard problem [25]. For example,

$$
\begin{aligned}
6s_1 + 7s_2 + 16s_3 + \ldots + 2s_n &\approx 3 mod 17\\
3s_1 + 6s_2 + 4s_3 + \ldots + 5s_n &\approx 16 mod 17\\
10s_1 + 4s_2 + 12s_3 + \ldots + 16s_n &\approx 12 mod 17\\
6s_1 + 10s_2 + 13s_3 + \ldots + 1s_n &\approx 3 mod 17\\
&\vdots\\
&\vdots\\
9s_1 + 5s_2 + 9s_3 + \ldots + 6s_n &\approx 9 mod 17\\
14s_1 + 15s_2 + 5s_3 + \ldots + 2s_n &\approx 8 mod 17
\end{aligned}
$$

To find the solution from the above set of equation that is the secret vector $s = \{s_1, s_2, \ldots, s_n\}$ is a hard problem if the value of n is large. Let us define the LWE and ring-LWE problem mathematically.

Definition 1 (*LWE*) Select $q \geq 2$, an integer $n \geq 1$, and an *error* $e \leftarrow \chi$ where, $e \in \mathbb{Z}_q$. Let $A_{s,\chi}$ be an oracle which output of the form $(\mathbf{a}, \mathbf{a}.\mathbf{s} + \mathbf{e}) \in \mathbb{Z}_{\mathbf{q}}^{\mathbf{n}} \times \mathbb{Z}_{\mathbf{q}}$, here, addition operations are work over finite field $\mathbb{Z}_q$, the polynomial $\mathbf{a} \in \mathbb{Z}_{\mathbf{q}}^{\mathbf{n}}$ generated uniformly at random and $s \in \mathbb{Z}_q^n$. LWE problem states that the probability of finding the secret vector s, from the arbitrary number of sample from an oracle $A_{s,\chi}$ is negligible [25].

Definition 2 (*Ring-LWE*) Select two numbers d and q as same as we defined in the basic notation. Define R_q as the ring $\mathbb{Z}_q[x]/(x^n + 1)$ containing all polynomials over the field $\mathbb{Z}_q$. Ring-LWE problem states that the probability of finding the secret polynomial $s \in R_q$, from a given pair of the polynomial $(\mathbf{a}, \mathbf{b} = \mathbf{a}.\mathbf{s} + \mathbf{e}) \in \mathbf{R_q} \times \mathbf{R_q}$ is negligible. Where $\mathbf{a} \in \mathbf{R_q}$ is uniformly generated polynomial and e is an error vector chosen independently from $e \leftarrow \chi^{\mathbf{N}}$ over R_q [25].

3.3 Homomorphic encryption

HE allows us to perform private computations. We can perform operations like addition and multiplication on ciphertext without decrypting the ciphertext or revealing the secret key used for encryption. In the traditional non-HE, to perform any operation on the ciphertext, the entity performing computations must have the secret key to decrypt the encrypted data. After the decryption, the entity performs the operations on that data after that it applies encryption again on that data. But if the object in question is an untrusted one then it is not prudent to share the secret key with it. HE removes the requirement of the secret key, especially with untrusted third parties. In HE, the computations can be performed without decrypting the ciphertext and re-encrypting it again. Given the property of HE to evaluation on ciphertext without having to decode it.

Homomorphic cryptosystem is an encryption function which forms a homomorphism and preserves the group operations performed on the ciphertext. Either addition or multiplication is usually chosen for the group operation in the cryptosystem. If HE operation is under additive then it can be written as:

$$H(x+y) = H(x) \otimes H(y). \quad (1)$$

Similarly, if HE operation is under multiplicative then it can be written as:

$$H(x \cdot y) = H(x) \otimes H(y). \quad (2)$$

where, H denotes an encryption function, $\otimes$ denotes the operation depending on the used cipher, and x, y are plaintext messages. Hence, we can see that the desired operation is preserved over the plaintexts x and y without the need to decrypt the message and perform it. HE schemes can be divided into three types based on the number of the operation:

- *Partially Homomorphic Encryption* support only one operation, either addition or multiplication. Example of homomorphic over additive schemes is pallier cryptosystem [26] and Goldwasser–Micali [27]. The homomorphic over multiplicative schemes is the RSA cryptosystem [28].
- *Fully Homomorphic Encryption* these schemes support both operations multiplication and addition simultaneously infinite number of times. The FHE schemes are defined over rings.
- *Somewhat homomorphic encryption* (SHE) schemes are similar to the FHE schemes; however, SHE supports two operations multiplication and addition simultaneously but, the number of operation is limited.

Pötzelsberger [29] points out that, SHE used in several applications, even though they perform operations a limited number of times, they provide much more efficiency and shorter ciphertext than fully homomorphic cryptosystems.

4 Leveled-BGV fully homomorphic encryption

The BGV is a Leveled-FHE scheme [15], follows Gentry's FHE scheme by employing lattice-based cryptography with other improvements. It does not rely on bootstrapping but instead applies it as an optimization. In this scheme, used parameters depend on the evaluation of the depth of the circuits; hence, it is known as a *leveled* scheme. The BGV scheme has significantly improved the performance compared to earlier proposed FHE schemes, which were based on bootstrapping. Also, the security assumption employed by BGV scheme is weaker compared to earlier ones. Ring-LWE forms the basis for the security of the BGV scheme, and it guarantees 2^{λ} security against known attacks. λ is a security parameter.

4.1 Basic scheme

The basic scheme for the BGV FHE is given below.

- Select the parameters: q, d, χ, and $\mathbf{N} = (2n+1)\log_2 q$, where, $n \in \mathbb{N}$.
- KeyGen: Sample $s' \in R_q^n$ and set $sk = s = (1, s')$. Generate a matrix uniformly at random $B \in R_q^{\mathbf{N} \times n}$, error vector $e \leftarrow \chi^{\mathbf{N}}$, and compute $b = Bs' + 2e \in R_q^{\mathbf{N}}$. Set $A = [b, -B] \in R_q^{\mathbf{N} \times (n+1)}$.
  Output: $(sk, pk) = (s, A)$
- Enc(m, pk): for message $m \in R_2$, set $\mathbf{m} = (\mathbf{m}, \mathbf{0}, \ldots, \mathbf{0}) \in \mathbf{R_2^{n+1}}$. Sample $r \in R_2{}^{\mathbf{N}}$ and set $c = \mathbf{m} + \mathbf{A^T r} \in \mathbf{R_q^{n+1}}$.
  Remark: Observe that $A.s = 2e$.
  Output ciphertext: c
- Dec(c, sk): computes $m = ((c.s)\ mod\ q) mod\ 2)$.
  Output: m
- *Correctness*

$$(((c.s) mod\ q) mod\ 2) = ((((m^T + r^T A).s) mod\ q) mod\ 2)$$

$$= (((m + 2r^T e) mod\ q) mod\ 2) = ((m + 2r^T e) mod\ 2) = m.$$

4.2 Noise management and modulus switching

In Gentry's first FHE scheme [6], we see that upon performing computations on the ciphertext, the noise gradually starts increasing and the ciphertext becomes more noise. Individually, performing one multiplication roughly square the noise whereas addition between two ciphertexts doubles the noise roughly. If noise in the ciphertext extends above the threshold level, then the ciphertext gives the wrong decryption and returns the incorrect value of the plaintext. The BGV scheme makes use of a keeps the noise in check by using a noise reduction technique which does not use bootstrapping, but rather reduces noise after performing each HE performed on the ciphertext. This technique which helps in noise reduction is called the *modulus switching* [14]. The lemma that describes the original modulus-switching technique says that

> An evaluator, who does not know the secret key s, but instead only knows bound on its length, can transform a ciphertext c modulo q into a different ciphertext c' modulo p while preserving correctness, expressed as $\langle c', s\rangle mod\ p \equiv (\langle c, s\rangle mod\ q)\ mod\ 2$[14].

where, $\langle,\rangle$ denotes Euclidean dot product. The interesting property found here is that the ciphertext actually decreases given the condition that the size of secret vector s is small and $p << q$. Therefore, the "noise magnitude" satisfy the condition $|\langle c', s\rangle \ mod \ p| < |\langle c, s\rangle \ mod \ q|$. The BGV scheme takes the technique above and further improves noise management by making use of a *ladder of gradually decreasing modulo*. Hence, BGV reduces the size of the noise without using the private key, besides, without the use of bootstrapping.

4.3 HElib

This BGV scheme has been implemented as a software library in C++, called HElib [19]. The implementation of the library focuses on the optimizations given by Gentry–Halevi–Smart, and also the ciphertext packing techniques are presented Smart–Vercauteren for improving the performance. In [20], described the algorithms used in the implementation of HElib and some considerations to have in mind depending on the hardware used.

5 Schulze voting method

Markus Schulze developed the Schulze voting method in [11]. This voting system produces output as a single winner among all the candidates using votes of the preference order of the voter. A sorted list of winners can also be obtained by using the Schulze voting method, in other words at the end of the voting result the rank obtained by each of the candidates. The Schulze method is also famous by other names, like "clone proof Schwartz sequential dropping" (CSSD), "Schwartz sequential dropping" (SSD), "path voting," "beat path method," "beat path winner," and "path winner." In the Schulze method, the winning candidate beats all the other candidates by a majority if we compare each candidate in a pairwise manner. Besides, there is also a chance that a candidate which beats all of its other competitors, if he/she compared pairwise to them in a pairwise fashion, then that candidate will win. Under the above properties, the Schulze method is defined as a Condorcet method. It is noted that this Condorcet method not provided to us by other preferential voting systems like instant-runoff and Borda. We get an ordered ranking from the Schulze voting method. If there are P number of vacant positions to be filled, then we can directly use the outcome of the Schulze voting method to fill the positions by choosing the top P positions. Many different ways have been proposed to implement the Schulze method, path method, and the Schwartz method are the most important ones. This voting system is being used by a lot of organizations including Ubuntu, Pirate Party, Debian, and many others.

5.1 Description of Schulze preferential ballot

In the Schulze method, each of the voters casts a vote where each vote has a list of the candidates' preference where he/she ranks the candidates' in order of preference, this way of input is same for all ranked single-winner electoral systems. In Table 2, we can see how the voter has to allot his preference to the candidates. Each vote cast has a list of all the candidate's name, and the voter puts his rank of preference besides the candidate's name. For example, let a voter whose name is Shyam, votes according to his preference, he has given preference "2" for his the first most favorite candidate Smith, "4" for his second-most favorite candidate Bob, "5" for his third-most favorite candidate Sony, and " 7" for his fourth-most favorite candidate Alice. In this voting method, all the voters follow the following rules

- Each voter may show indifference between two or more candidates by giving the same rank or preference to them.
- Each voter may use the non-consecutive number to express his preference as it is immaterial in our case. If the number is higher, then we know that the candidate is less preferred than a candidate with a lesser number.
- If a voter doesn't give any candidate the preference number, then it is considered that the voter (a) is indifferent among the unranked candidates (b) prefers all the ranked candidates over unranked candidates.

5.2 Mathematical construction of Schulze voting method

We assume that the total number of candidate participate in the election is N. We denote $\delta[P, Q]$ as the total number of voters who have given the rank of P higher than the rank of Q. Where, P, Q are the participating candidates. A path P to Q is a sequence of candidates $\kappa(1), \ldots, \kappa(N)$ which satisfy the following characteristic:

- $\kappa(1) = P$ and $\kappa(N) = Q$.
- For
  $j = 1$ to $(N-1) : \delta[\kappa(j), \kappa(j+1)] > \delta[\kappa(j+1), \kappa(j)]$

The above mathematical expression says that the candidate P, beats all the participating candidates, if we perform the pairwise comparison between each candidate then there exists a unique path for which above mathematical expression satisfies the condition. The following mathematical backgrounds are used in the computation of the Schulze voting method.

Table 2 Example of Schulze preferential ballot

Name of the candidate	Voter preferences
Alice	7
Bob	4
Smith	2
Sony	5

Instruction for the voter please rank any number of options in your order of preference

- If there exists a path in the sequence of comparisons from candidate P to candidate Q which satisfy the following expression $\delta[\kappa(j), \kappa(j+1)] \geq \rho$, where $j = 1$ to $(N-1)$, then we called ρ is a strength of the path or smallest number of voters in the sequence of comparisons.
- If there is at least one path between the candidates P and Q, then the maximum strength $\rho[P, Q]$ is called the strength of the strongest path connecting between them. We will write $\rho[P, Q] = 0$ if there exists no path between the candidates P and Q.
- If $\rho[P, Q] > \rho[Q, P]$ then we call it candidate P *beats* the candidate Q.
- If $\rho[P, Q] \geq \rho[Q, P]$ for every other candidate Q, then P is a *potential winner*.
- If $\rho[P, Q] > \rho[Q, P]$ and $\rho[Q, R] > \rho[R, Q]$ are true, then it is $\rho[P, R] > \rho[R, P]$ also true. Therefore, it is ensured.
  - that the above term *beats* satisfy the rule of transitive relation while comparing the strengths of two candidates.
  - that there is always a unique candidate P which holds $\rho[P, Q] \geq \rho[Q, P] \; \forall \; Q \in \kappa$, where κ is the set of all the candidates excluding P.

5.3 Example

We have taken an example from [30], for the better understanding of Schulze voting method. Let $|V|$ denotes the number of voters and $|C|$ denotes the total number of candidates. Besides, we also assume that $|V| = 45$ and $|C| = 5$. For simplicity, we consider that the name of five candidates are A, B, C, D, and E. In Table 3, we have represented the pairwise preferences of the voters. First of all, we compute the pairwise preference of the voters from this Table 3.

For instance, if we compared pairwise preference between the candidate C and D, then we found that there are $5 + 7 + 3 + 2 = 17$ voters who favour C to D and $5 + 8 + 7 + 8 = 28$ voters who favour D to C. Therefore, $\delta[\mathrm{C}, \mathrm{D}] = 17$ and $\delta[\mathrm{D}, \mathrm{C}] = 28$. Using this method, we create a matrix for pairwise comparisons among all the candidates is given in Table 4.

To find the winner of the candidates, we compute the strongest path in the following way: first of all, we construct a directed graph from the pairwise preferences Table 4. The constructed directed graph is given in Fig. 2. In this directed graph, the number of vertices is equal to the total number of participating candidates in the election. Therefore, we named each vertex as a candidate name. The weight of the direct edge from a candidate R to S is denoted as $\delta[R, S]$. To provide an uncluttered and clean directed graph, we use only those edges from R to S when it satisfies the condition $\delta[R, S] > \delta[S, R]$. Where R, S denotes set of all candidates.

If we calculate the strongest path between the two candidates A and D, then we found that $\rho[\mathrm{A}, \mathrm{D}] = 30$. Hence, the direct edge A to D is the strongest path whose strength is 30, as shown in Fig. 3.

If we compute the strongest path between the candidates E and B, then we found that the direct edge (E, B) of strength 27 is not the strongest path from E to B, in place of the strongest path is the alternative edge ($\mathrm{E} \rightarrow \mathrm{D} \rightarrow \mathrm{C} \rightarrow \mathrm{B}$) which has strength $\min(31, 28, 29) = 28$ as shown in Fig. 4. Hence, the value of $\rho[\mathrm{E}, \mathrm{B}] = 28$. Similarly, when if we look at all the pairwise paths, then we get the following strengths of the strongest path as shown in the Table 5. We can now look at the output of the Schulze method in the following way. If compare the strength between candidate A and B, then we found that $\rho[\mathrm{A}, \mathrm{B}] > \rho[\mathrm{B}, \mathrm{A}]$, i.e., in a pairwise comparison the candidate A beats B. Another example is that $\rho[\mathrm{B}, \mathrm{D}] > \rho[\mathrm{D}, \mathrm{B}]$, so candidate B beats the candidate D. Hence, from the rule of transitivity, we can say that candidate A beats both the candidate B and D. Hence, if we keep on doing this, then we get the order of the final sequence $\mathrm{E} > \mathrm{A} > \mathrm{C} > \mathrm{B} > \mathrm{D}$. Since, E beats every other candidate, so he/she is the winner of the candidates.

6 Private computation of Schulze voting method

In our proposed work, we make the following assumptions and also limit our scope to particular aspects only.

- We don't consider how the election will take place, for simplicity we will assume the traditional client and cloud architecture where the preference matrix will be outsourced in an encrypted manner to the service provider which is not completely trusted. The computing entity will privately compute the algorithm to find the strongest path and return the result to the client.
- We assumed that when all the voters provide their preferences in the ballot then after the end of voting, the

Table 3 Pairwise preference of voters

Number of voters who have given the same preference	Order of preference
7	DCEBA
5	ACBED
8	BEDAC
2	CBADE
3	EBADC
3	CABED
5	ADECB
7	CAEBD

Table 4 Matrix for preference table

	$\delta[\star, A]$	$\delta[\star, B]$	$\delta[\star, C]$	$\delta[\star, D]$	$\delta[\star, E]$
$\delta[A, \star]$		20	26	30	22
$\delta[B, \star]$	25		16	33	18
$\delta[C, \star]$	19	29		17	24
$\delta[D, \star]$	15	12	28		14
$\delta[E, \star]$	23	27	21	31	

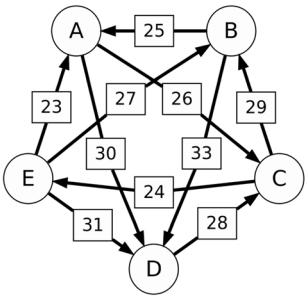

Fig. 2 Directed graph representing voting preferences

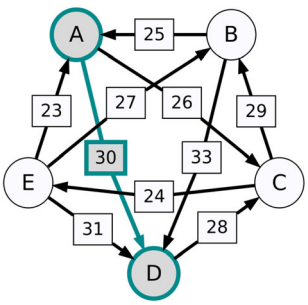

Fig. 3 The strongest path between A and D

electronic voting machine automatically creates a matrix of pairwise preference in encrypted form.

The process of creation of an encrypted matrix of pairwise preference from an unencrypted matrix of the pairwise preference system model is given in Fig. 5. In order to find the winner of the candidates privately, the Schulze voting method requires the computation of the strongest path in a

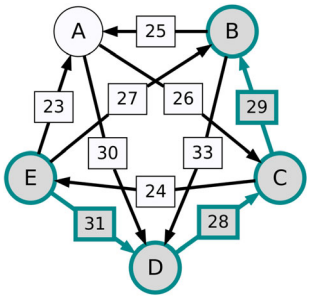

Fig. 4 The strongest path between E and B

Table 5 Matrix for the strongest path

	$\delta[\star, A]$	$\delta[\star, B]$	$\delta[\star, C]$	$\delta[\star, D]$	$\delta[\star, E]$
$\delta[A, \star]$		28	28	30	24
$\delta[B, \star]$	25		28	33	24
$\delta[C, \star]$	25	29		29	24
$\delta[D, \star]$	25	28	28		24
$\delta[E, \star]$	25	28	28	31	

private manner. For the computation of the strongest path in a private way requires comparison, equality, maximum, and minimum of two encrypted numbers. Therefore, before we go to explain the private computation of the Schulze voting method, we need first to discuss these terminologies one by one.

6.1 Evaluation over ciphertext

6.1.1 Comparison

Let us consider two integers α and β that we want to compare represented in binary form as $\alpha = \overline{\alpha_0 \alpha_1 \ldots \alpha_{n-1}}$ and $\beta = \overline{\beta_0 \beta_1 \ldots \beta_{n-1}}$. In order to compare two numbers, we first compare the MSB, if the MSB of α is greater than the MSB of β, then α is greater than β. If the MSB of both α and β are equal, then we compare the next bit, and so on. Let us take a concrete example to understand the same. Suppose we have two 1 bit numbers $\alpha = \alpha_0$ and $\beta = \beta_0$. From the truth Table 6 we can see that if $\alpha_0 > \beta_0$ then $c_0 = \alpha_0.\beta_0' = 1$

Table 6 Comparison of one-bit number

α_0	β_0	$\alpha_0.\beta_0'$
0	0	0
0	1	0
1	0	1
1	1	0

and if $\alpha_0 \leq \beta_0$ then $c_0 = \alpha_0.\beta_0' = 0$, where " ′ " denote the complement.Similarly, we consider that $\alpha = (\alpha_0, \alpha_1)$ and $\beta = (\beta_0, \beta_1)$ are two-bit numbers. If $\alpha > \beta$ then the expression $c_1 = (\alpha_0\beta_0^{'} \oplus (1 \oplus \alpha_0 \oplus \beta_0).\alpha_1\beta_1^{'})$ equal to 1, where "." is the AND or multiplication operator. Similar logic can be extended for n bit as $c_{n-1} = c_{n-2} \oplus (\alpha_{n-2} \oplus \beta_{n-2} \oplus 1).\alpha_{n-1}.\beta_{n-1}^{'}$. If we perform multiplication and addition which convert to AND and XOR operations in $\mathbb{Z}_2$ in HElib library. We proposed an algorithm COM for the comparison of two integers in algorithm 1. We can also compare two encrypted integers by using FHE.

- COM(**Enc**(α), **Enc**(β)) = C. (C is an encrypted value)
- **Dec**(C) = 1 if $\alpha > \beta$
- **Dec**(C) = 0 otherwise.

Hence, we can define a function

$$\mathrm{COM}(\alpha, \beta) = \begin{cases} 1 & \alpha > \beta \\ 0 & \alpha \leq \beta \\ Enc(1) & Enc(\alpha) > Enc(\beta) \\ Enc(0) & Enc(\alpha) \leq Enc(\beta) \end{cases}$$

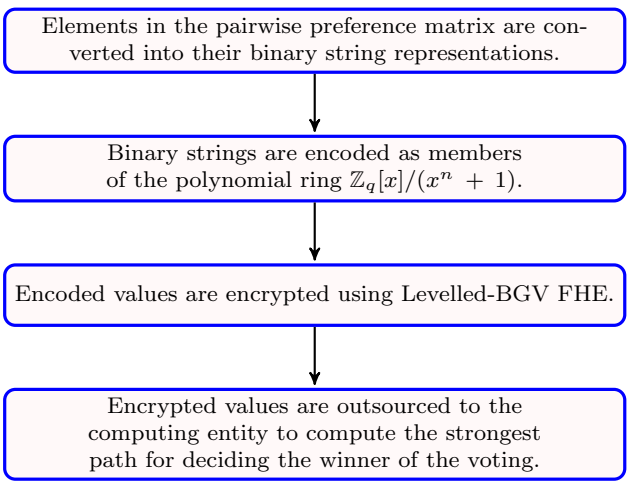

Fig. 5 System model for private computation of Schulze voting method

Algorithm 1 COM(α, β)

Input : (α, β)
Output : 0 or 1

1: $\alpha = \overline{\alpha_0\alpha_1 \cdots \alpha_{n-1}}$
2: $\beta = \overline{\beta_0\beta_1 \cdots \beta_{n-1}}$
3: $c_0 = \alpha_0.\beta_0^{'}$
4: **for** $i = 1$ to $n - 1$ **do**
5: $\quad c_i = c_{i-1} \oplus (\alpha_{i-1} \oplus \beta_{i-1} \oplus 1).\alpha_i.\beta_i^{'}$
6: **end for**
7: **return** c_i

Listing 1: Code for compare function

```
inline Ctxt compare2(Ctxt& B,Ctxt& A,
EncryptedArray &ea,Ctxt &cZeroOne,
Ctxt &cOneZero, Ctxt &cOne,long nslots)
{
Ctxt cAB(A.getPubKey());
cAB = A;
cAB.multiplyBy(B);
Ctxt cX(A.getPubKey());
cX = A;
NOT(cX,cOne);
Ctxt nB(B.getPubKey());
nB = B;
NOT(nB,cOne);
cX.multiplyBy(nB);
cX.addCtxt(cAB);
Ctxt anB(A.getPubKey());
anB = A;
anB.multiplyBy(nB);
//return anB;
//result = cOneZero;
int iter;
//return cX;
vector<Ctxt> cXShiftladder;
cXShiftladder.push_back(anB);
ea.shift(cX,1);
cX.addCtxt(cZeroOne);
cXShiftladder.push_back(cX);
for(iter = 1 ; iter < nslots - 1; iter++){
Ctxt nC(cX.getPubKey());
nC = cXShiftladder[cXShiftladder.size() - 1];
ea.shift(nC,1);
nC.addCtxt(cZeroOne);
cXShiftladder.push_back(nC);
}
Ctxt result(A.getPubKey());
result = mulOrdiVector(cXShiftladder,ea,nslots);
vector<Ctxt> orArray;
orArray.push_back(result);
for(iter = 0 ; iter < nslots - 1 ; iter++){
Ctxt nC(orArray[0].getPubKey());
nC = orArray[orArray.size() - 1];
ea.shift(nC,1);
orArray.push_back(nC);
}
Ctxt returnResult(orArray[0].getPubKey());
returnResult = orVector(orArray,ea,cOne,nslots);
cout << "Done Compare" << endl;
return returnResult;
```

In listing 1, we present the code for our compare function as written in HElib.

6.1.2 Equality

We consider the method for computing equality between two numbers, which return 1 if the numbers are the same and 0 otherwise. In order to understand the equality between two number, we take a concrete example. Suppose we have two 1 bit numbers $\alpha = \alpha_0$ and $\beta = \beta_0$. In the Table 7, we can see that if $\alpha_0 = \beta_0$ then the Boolean expression $\alpha_0 \oplus \beta_0 \oplus 1$ is equal to 1 and if $\alpha_0 \neq \beta_0$ then the Boolean expression $Z_0 = \alpha_0 \oplus \beta_0 \oplus 1$ is equal to 0. Similarly, we consider that $\alpha = (\alpha_0, \alpha_1)$ and $\beta = (\beta_0, \beta_1)$ are two-bit numbers. If $\alpha = \beta$ then the expression $Z_1 = (\alpha_0 \oplus \beta_0 \oplus 1).(\alpha_1 \oplus \beta_1 \oplus 1)$ equal to 1. Similar logic can be extended for n bit as $Z_{n-1} = \prod_{i=0}^{n-1}(\alpha_i \oplus \beta_i \oplus 1)$. When we perform addition and multiplication over ciphertext, then it effects similar in plaintext because of addition and multiplication in ciphertext behave similarly to XOR and AND operations respectively. We propose an algorithm EQU for the equality of two integers in algorithm 2. We can also perform an equality check between two encrypted integers by using FHE.

- EQU($\mathbf{Enc}(\alpha)$, $\mathbf{Enc}(\beta) = C$. (C is an encrypted value)
- $\mathbf{Dec}(C) = 1$ if $\alpha = \beta$
- $\mathbf{Dec}(C) = 0$ otherwise.

Hence, we define a functionHence, we define a function that check equality of two bits

$$\mathrm{EQU}(\alpha, \beta) = \begin{cases} 1 & \alpha = \beta \\ 0 & \alpha \neq \beta \\ Enc(1) & Enc(\alpha) = Enc(\beta) \\ Enc(0) & Enc(\alpha) \neq Enc(\beta) \end{cases}$$

Algorithm 2 EQU(α,β)

Input : (α, β)
Output : 0 or 1

1: $\alpha = \overline{\alpha_0\alpha_1 \cdots \alpha_{n-1}}$
2: $\beta = \overline{\beta_0\beta_1 \cdots \beta_{n-1}}$
3: $Z = 1$
4: **for** $i = 0$ **to** $n - 1$ **do**
5: $\quad Z = Z \times (\alpha_i \oplus \beta_i \oplus 1)$
6: **end for**
7: **return** Z

Table 7 Equality check between one-bit number

α_0	β_0	$\alpha_0 \oplus \beta_0 \oplus 1$
0	0	1
0	1	0
1	0	0
1	1	1

Listing 2: Code for equality function

```
inline Ctxt isEqual2(Ctxt B,Ctxt A,Ctxt
    cOne,EncryptedArray ea,long nslots){
Ctxt result(A.getPubKey());
result = A;
result.addCtxt(cOne);
result.addCtxt(B);
//  A.multiplyBy(B);
//   B.addCtxt(cOne);
//   result.multiplyBy(B);
//   result.addCtxt(A);
vector<Ctxt> shiftArray;
shiftArray.push_back(result);
long iter;
for(iter = 0 ; iter < nslots - 1 ; iter++){
Ctxt nC(result.getPubKey());
ea.rotate(result,1);
nC = result;
shiftArray.push_back(nC);
}
Ctxt resultReturn =
    mulOrdiVector(shiftArray,ea,nslots);
return resultReturn;
}
```

In listing 2, we present the code for equality function written in HElib.

6.1.3 Maximum

Now, We will describe a function MAX (), which find the maximum value of two encrypted number with the help of function EQU () and COM (), which we have already defined in the previous subsections. The detailed description of MAX algorithm is given in Algorithm 3.

Algorithm 3 MAX(α, β)

Input : (α, β)
Output : maximum of (α, β)

1: Z = COM(α, β).α + COM(β, α).β + EQU(α, β).α
2: **return** Z

In order to find the maximum value of MAX (α, β), we have taken an example. Consider that if $\alpha = 5$ and $\beta = 7$ then

$$\mathrm{MAX}(5,7) = \mathrm{COM}(5,7).5 + \mathrm{COM}(7,5).7 + \mathrm{EQU}(5,7).5 = 0.5 + 1.7 + 0.5 = 7$$

. We will get a similar result when we execute the MAX() algorithm which has input in the form of ciphertext. Therefore, we define a function

$$\mathrm{MAX}(\alpha, \beta) = \begin{cases} \alpha & \alpha \geq \beta \\ \beta & \beta > \alpha \\ Enc(\alpha) & Enc(\alpha) \geq Enc(\beta) \\ Enc(\beta) & Enc(\beta) > Enc(\alpha). \end{cases}$$

6.1.4 Minimum

Similar to the maximum function, we describe a function $\text{MIN}(\alpha, \beta)$ which find the minimum value between α and β with the help of function $\text{EQU}(\alpha, \beta)$ and $\text{COM}(\alpha, \beta)$ which we have already defined in the previous subsections. The description of the MIN algorithm given in Algorithm 4. We have taken $\alpha = 5$ and $\beta = 7$ as an example, to describe $\text{MIN}(\alpha, \beta)$ algorithm. $\text{MIN}(5, 7) = \text{COM}(5, 7).7 + \text{COM}(7, 5).5 + \text{EQU}(5, 7).5 = 0.7 + 1.5 + 0.5 = 5$. We will get a similar result when we execute the MIN() algorithm which has input in the form of ciphertext. Therefore, we define a function

Algorithm 4 $\text{MIN}(\alpha, \beta)$

Input : (α, β)
Output : minimum of (α, β)
1: $\text{Z} = \text{COM}(\alpha, \beta).\beta + \text{COM}(\beta, \alpha).\alpha + \text{EQU}(\alpha, \beta).\alpha$
2: **return** Z

$$\text{MIN}(\alpha, \beta) = \begin{cases} \alpha & \alpha \leq \beta \\ \beta & \beta < \alpha \\ Enc(\alpha) & Enc(\alpha) \leq Enc(\beta) \\ Enc(\beta) & Enc(\beta) < Enc(\alpha) \end{cases}$$

6.1.5 Addition

We have used a binary circuit adder to add two encrypted numbers which have been already implemented in HElib library, as shown in listing 3.

Listing 3: Code for binary carry adder

```
inline Ctxt addMemo(Ctxt& B,Ctxt& A,EncryptedArray&
    ea,long nslots){
  Ctxt cTerm(A.getPubKey());
  //cTerm = A;
  long iter;
  for(iter = 0 ; iter < nslots ; iter++){
     cTerm = A;
     B.addCtxt(A); // Addition in B
     cTerm.multiplyBy(B); // carry in C
     A.addCtxt(cTerm);
     ea.shift(A,-1);
  }
  return B;
}
```

6.2 Private computation of the strongest path strength

The most challenging part of implementing the private computation of Schulze voting method is private computation of the strongest path strength. In this section, we propose the algorithm for private computation of strongest path with the help of COM(), EQU() and MAX(), MIN() function as described in the previous subsections. We have proposed an algorithm for private computation of the strongest path strength in Algorithm 5. This Algorithm 5 is executed by the cloud server after receiving the encrypted matrix of pairwise preference from the client. We can see that in Algorithm 5 of line 4 requires a comparison of two encrypted data. Therefore, the cloud server applies COM algorithm for the comparison of two encrypted data. Besides, in Algorithm 5, we see that line number 13 and 14 require finding the minimum and maximum value of two encrypted data, respectively. So, in order to get the correct result, the cloud server executes MIN and MAX algorithm, as we defined in the previous subsection. After the final execution of Algorithm 5, the cloud server gets the encrypted matrix of the strongest path. After that, the cloud server finds the winner of the candidate using that encrypted matrix. For simplicity, we have considered the previous example 5.3 of Schulze voting method. The cloud server compares the strength between candidate A and B using MAX function, if $\text{MAX}(\rho[\text{A}, \text{B}], \rho[\text{B}, \text{A}])$ produces output $\rho[\text{A}, \text{B}]$, i.e., in a pairwise comparison the candidate A beats B. Another example is that if $\text{MAX}(\rho[\text{B}, \text{D}], \rho[\text{D}, \text{B}])$ produces output $\rho[\text{B}, \text{D}]$, then the candidate B beats the candidate D. Therefore, from the rule of transitivity, the cloud server can say that candidate A beats both the candidate B and D. Hence, if the cloud server keep on doing this, then he/she gets the order of the final sequence. The cloud server sends the encrypted result of the winner to the client. The client declares the actual winner of the candidate after decryption of the encrypted result using their own secret key.

Algorithm 5 Strongest Path Strength(δ)

Input : Encrypted matrix of pairwise preference δ
Output : Encrypted matrix of strongest path ρ
1: **for** $i = 0$ to $N - 1$ **do**
2: **for** $j = 0$ to $N - 1$ **do**
3: **if** $i \neq j$ **then**
4: $\rho[i, j] = \text{COM}(\delta[i, j], \delta[j, i]).\delta[i, j]$
5: **end if**
6: **end for**
7: **end for**
8: **for** $i = 0$ to $N - 1$ **do**
9: **for** $j = 0$ to $N - 1$ **do**
10: **if** $i \neq j$ **then**
11: **for** $k = 0$ to $N - 1$ **do**
12: **if** $i \neq k$ and $j \neq k$ **then**
13: $temp = \text{MIN}(\rho[j, i], \rho[i, k])$
14: $\rho[j, k] = \text{MAX}(\rho[j, k], temp)$
15: **end if**
16: **end for**
17: **end if**
18: **end for**
19: **end for**
20: **return** ρ

6.3 Real-world scenario

In an election, there may be a large number of polling stations with electronic voting machines. An electronic voting machine a light-weighted device that cannot perform heavy computation over the ciphertext required in homomorphic encryption. Moreover, the results of all electronic voting machines located at different polling stations need to be collated for obtaining the final result. The computational and collation requirements can be met by connecting all the machines to the cloud. Once the voting process is completed, the cloud server requests the encrypted matrix of the preference table from each voting machine and determines the winner of the candidate by computation in the encrypted domain. The cloud server sends the encrypted results of the winning candidate to the trusted third party. Using a private key, the trusted third party extracts the results and announces the winning candidate.

7 Results and implementation

In this section, we will analyze the time taken by the algorithm to run against the different parameters. In addition, we will analyze the time taken by the functions COM(), EQU(), ADD(), and MAX(). We will also plot a graph for the time taken by each of these functions and then calculate the time taken for the whole algorithm for the different number of candidates. The parameters used for the implementation of Schulze voting method in HElib library are given in the Table 8.

The following constant parameters, as shown in Table 9 have been used in HElib. We implemented each of the functions in C++ (gcc version 7.3.0) and deployed it on Intel(R) Core(TM) *i*5-3230*M* CPU clocked at 2.60GHz with 4 GB RAM and we make use of these to estimate the time taken by the Algorithm 5 for different parameters. Throughout the implementation of Schulze voting method, we have used of HElib library, which is based on the BGV FHE scheme.

Table 8 Meaning of parameters used for implementation in HElib

Parameters	Meaning
p	Characteristic of plaintext space
k	Security parameter
r	Lifting parameter
d	Embedding degree
c	Number of columns in the key-switching matrix
w	Hamming weight of the secret key
s	Minimum number of slots
m	Chooses the mth cylotomic polynomial.
L	Number of levels

Table 9 Parameters that are used in the implementation of HElib

Parameters	Values
Library	HElib
p	2
r	1
d	1
c	2
k	128
w	64
s	16
m	255

In the Fig. 6, we have plotted the graph of time taken by each of the functions for various number of levels also known as depth of circuit i.e. the length of modulus chain. From Fig. 6, it is clear that if we increase the depth of the circuit, then the computation time of the function also increases. Besides, it is also clear from the graph that, the computation time of the EQU() and Binary Carry Addition are approximately same, whereas computation time of the COM() is approximately double the time taken by the EQU() and Binary Carry Addition.

We have plotted the graph of time taken by the functions MAX() and MIN() for the various number of levels in Fig. 7. When we compute the running time of the functions MAX() and MIN(), then we found that the time taken by both the functions are the same. Besides, we can see that if we increase the depth of our circuit, then the computation time of the functions also increases.

In Figs. 8 and 9, we have plotted the graph of time taken by the computation of strongest path strength in the

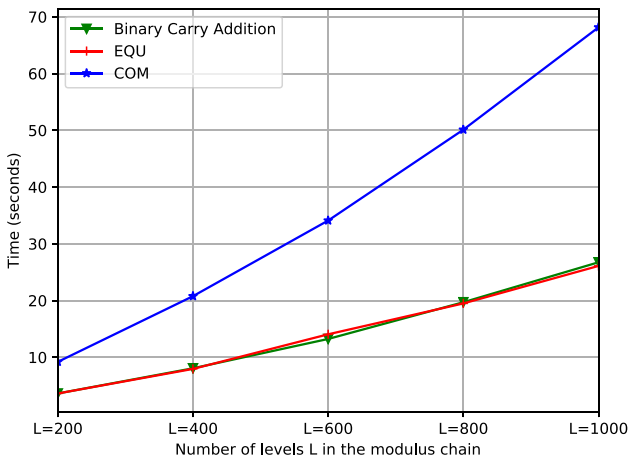

Fig. 6 Running time of the functions binary carry addition(ADD)(α, β), EQU(α, β), and COM(α, β)

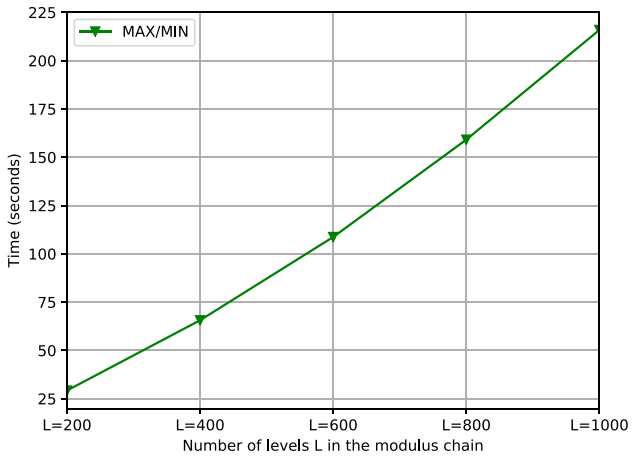

Fig. 7 Running time of the function MAX/MIN(α, β)

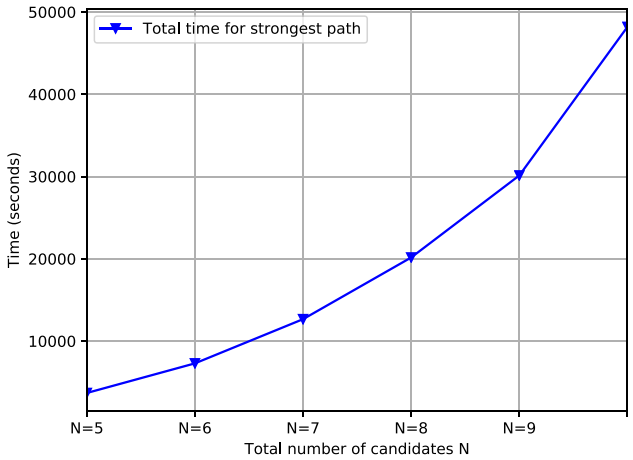

Fig. 8 Running time of the private computation of strongest path for $L = 200$

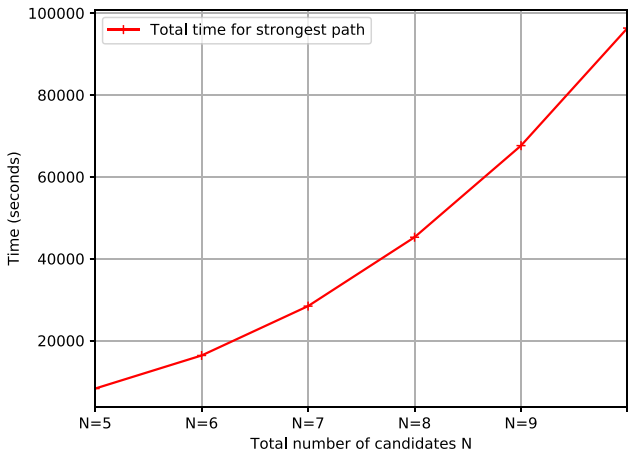

Fig. 9 Running time of the private computation of strongest path for $L = 400$

Table 10 Size of the public key and secret key for different values of L

L	Public key size (MB)	Secret key size (MB)
200	6.3	6.6
400	12.6	13.1
600	18.7	19.5
800	25.1	26.1
1000	31.2	32.6

Schulze voting method for the various number of candidates. We can see that, if we increase the number of candidates in the election, then the computation time of the strongest path is also increased substantially. From these figures, we can conclude that if we increase the number of levels, then the computation time is increased. Therefore, for efficient computation, we set the optimum value of L that compute Schulze voting method efficiently and correctly.

We have also computed the size of the secret key and the size of the public key used for encryption and decryption in the Table 10. From this table, it is also clear that if we increase the value of L, then the size of the public key and the size of the private key is also increases.

7.1 Time and space complexity

In our proposed scheme, the total number of encryption operations performed by the Electronic Voting Machine (EVM) is $N(N-1)$, and the total number of decryption operations performed by the EVM is also $N(N-1)$. According to BGV [15], if we do not use the bootstrapping, then the time complexity of the FHE scheme is $\mathcal{O}(\lambda.L^3)$ and if we use bootstrapping then the time complexity of the FHE scheme is $\mathcal{O}(\lambda^2)$, where λ is a security parameter. Therefore, the time complexity requires by the EVM is $\mathcal{O}(N(N-1)\lambda.L^3)$ (without bootstrapping) or $\mathcal{O}(N(N-1)\lambda^2)$ (with bootstrapping). To find the strongest path strength, the cloud server executes Algorithm 5. The time complexity of the Algorithm 5 is N times the time complexity of COM() function and N^3 times the time complexity of MIN() and MAX() functions. Each of the functions like COM(), MAX(), and MIN() requires $\mathcal{O}(n)$ time complexity. Therefore, overall time complexity requires by the cloud server is $\mathcal{O}(nN) + \mathcal{O}(2nN^3) \approx \mathcal{O}(nN^3)$. Therefore, overall time complexity of the our proposed scheme is $\mathcal{O}(n.N^3 + N.(N-1)\lambda.L^3)$ (without bootstrapping) or $\mathcal{O}(nN^3 + N.(N-1).\lambda^2)$ (with bootstrapping). According to BGV [15] space complexity of the

scheme is $\mathcal{O}(\lambda)$. The total number of ciphertext exchanges between the cloud server and the EVM is $N(N-1)$. Therefore overall space complexity of the proposed scheme is $\mathcal{O}(N.(N-1).\lambda)$.

8 Conclusion

We have provided a novel way to conduct Condorcet voting privately. We have presented an algorithm for computing the strongest paths in a directed weighted graph. Besides, we have provided an open-source code for the same for the research community to use. Through examples and actual implementations, this work presents the challenges and better understanding of computing on encrypted data using FHE. The paper also discussed the effect of different parameters like time taken and memory usage during computation over encrypted data. From the results of the execution time, we can conclude that if we increased the number of levels, then the computation time is also increased. Therefore, for efficient computation, we set the optimum value of L in the implementation so that it computes the Schulze voting method efficiently and correctly.

Funding None.

Compliance with ethical standards

Conflicts of interest The authors declare that he/she has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Aghili, S.F., Mala, H., Shojafar, M., Conti, M.: Pakit: Proactive authentication and key agreement protocol for internet of things. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 348–353. IEEE (2019)
2. Pooranian, Z., Chen, K.C., Yu, C.M., Conti, M.: Rare: Defeating side channels based on data-deduplication in cloud storage. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 444–449. IEEE (2018)
3. Imani, M., Ghoreishi, S.F., Braga-Neto, U.M.: Bayesian control of large mdps with unknown dynamics in data-poor environments. In: Advances in Neural Information Processing Systems, pp. 8146–8156 (2018)
4. Imani, M., Ghoreishi, S.F., Allaire, D., Braga-Neto, U.M.: Mfbo-ssm: multi-fidelity bayesian optimization for fast inference in state-space models. Proc. AAAI Conf. Artif. Intell. **33**, 7858–7865 (2019)
5. Thillaiarasu, N., ChenthurPandian, S.: A novel scheme for safeguarding confidentiality in public clouds for service users of cloud computing. Clust. Comput. **22**(1), 1179–1188 (2019)
6. Gentry, C., Boneh, D.: A Fully Homomorphic Encryption Scheme, vol. 20. Stanford University, Stanford (2009)
7. Prevost, J.J.: A comprehensive solution for research-oriented cloud computing. In: Cloud Computing–CLOUD 2018: 11th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25–30, 2018, Proceedings, vol. 10967, p. 407. Springer, New York (2018)
8. How counting of votes is done in india. https://www.mapsofindia.com/my-india/india/election-results-2017-how-counting-of-votes-is-done-in-india. Accessed 15 Apr 2018
9. Sandler, D., Derr, K., Wallach, D.S.: Votebox: a tamper-evident, verifiable electronic voting system. In: USENIX Security Symposium, vol. 4, p. 87 (2008)
10. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 539–556. Springer, New York (2000)
11. Schulze, M.: A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. Soc. Choice Welf. **36**(2), 267–303 (2011)
12. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: International Workshop on Public Key Cryptography, pp. 420–443. Springer, New York (2010)
13. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 24–43. Springer, New York (2010)
14. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Annual Cryptology Conference, pp. 505–524. Springer, New York (2011)
15. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory (TOCT) **6**(3), 13 (2014)
16. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, pp. 1219–1234. ACM (2012)
17. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference, pp. 75–92. Springer, New York (2013)
18. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. J. Cryptol. (2019). https://doi.org/10.1007/s00145-019-09319-x
19. Halevi, S., Shoup, V.: HElib: an implementation of homomorphic encryption. https://github.com/shaih/HElib. Accessed 15 Feb 2014
20. Shoup, V., Halevi, S.: Algorithms in helib. In: Annual Cryptology Conference, pp. 554–571. Springer, New York (2014)
21. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 617–640. Springer, New York (2015)
22. Kaur, H., Sood, S.K., Bhatia, M.: Cloud-assisted green iot-enabled comprehensive framework for wildfire monitoring. Clust. Comput. 1–14 (2019)
23. Huang, X., Li, C., Chen, H., An, D.: Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. Clust. Comput. 1–11 (2019)

24. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC '05, pp. 84–93. ACM (2005)
25. Regev, O.: The learning with errors problem (invited survey). In: 2010 IEEE 25th Annual Conference on Computational Complexity, pp. 191–204 (2010). https://doi.org/10.1109/CCC.2010.26
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT'99, pp. 223–238. Springer, New York (1999)
27. Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC '82, pp. 365–377. ACM (1982)
28. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Found. Secur. Comput. **4**(11), 169–180 (1978)
29. Pötzelsberger, G.: Kv web security: applications of homomorphic encryption. http://www.fim.uni-linz.ac.at/Lva/Web_Security/Abgaben/Poetzelsberger-Homomorphic.pdf. Accessed 23 May 2013
30. Contributors, W.: Schulze method. https://en.wikipedia.org/wiki/Schulze method. Accessed 5 Nov 2018

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Vijay Kumar Yadav received the B.Tech. degree in Information Technology from the Aryabhatta Knowledge University, Patna, India, and the M.Tech. degree in Cyber Law and Information Security from the Indian Institute of Information Technology, Allahabad, India, where he is currently pursuing the Ph.D. degree. His research interest areas are cryptography, information security and security in wireless sensor networks.

Anshul Anand received the B.Tech. degree in Information Technology from the Indian Institute of Information Technology, Allahabad, India, and the M. Tech. degree in Cyber Law and Information Security from the Indian Institute of Information Technology, Allahabad, India. His research interest areas are cryptography, information security and security in wireless sensor networks.

Shekhar Verma received the Ph.D. degree in computer networks from the Indian Institute of Technology, Varanasi, India. He is currently a Professor (IT) with the Indian Institute of Information Technology, Allahabad, India. His research interest areas are machine learning, computer networks, wireless sensor networks, wireless networks, information and networks security, vehicular technology, and cryptography.

S. Venkatesan received the Ph.D. degree in Mobile Agent Security from the Anna University, Chennai, Tamilnadu, India. He is currently a Assistant Professor (IT) with the Indian Institute of Information Technology, Allahabad, India. His research interest areas are mobile agent security, cryptography, block chain, cloud computing and social network privacy.